Es. 12.1 – PuzzleBobble

Un `PuzzleBobble` rappresenta una versione semplificata del celebre videogioco. Lo schema di gioco è composto da una matrice di 6 colonne e 10 righe di caselle. Ogni casella può essere libera oppure occupata da una bolla, che a sua volta può essere di vari colori. Implementare le seguenti operazioni che possono essere effettuate su un `PuzzleBobble`:

--- Metodi invocati nella PRIMA PARTE di main.cpp: ---

✓ **`PuzzleBobble pb;`**

Costruttore di default che inizializza un `PuzzleBobble` inizialmente vuoto di bolle.

✓ **`cout << pb;`**

Operatore di uscita per il tipo `PuzzleBobble`. Lo schema viene stampato secondo il seguente formato:

```
========
|RRBY G|
|R RR G|
|Y     |
|      |
|      |
|      |
|      |
|      |
|      |
========
```

I bordi dello schema sono rappresentati da caratteri `'='` e `'|'`. Una casella vuota è rappresentata da uno spazio, mentre una bolla è rappresentata dalla lettera maiuscola corrispondente al suo colore: `'R'` per rosso, `'G'` per verde, `'B'` per blu, `'Y'` per giallo.

✓ **`pb.fire(i,color);`**

Funzione che spara una bolla di colore `color` dal bordo inferiore dello schema lungo la colonna di indice `i`. Il colore è specificato da uno dei caratteri maiuscoli `'R'`, `'G'`, `'B'`, `'Y'`. L'indice `i` della colonna è un numero da 0 (prima colonna a sinistra) a 5 (ultima colonna a destra). La bolla scorre dal basso verso l'alto e si ferma "attaccandosi" sotto la prima bolla che trova su quella colonna, oppure sotto al limite superiore dello schema se non trova bolle. Per esempio, se nello schema visualizzato sopra viene chiamata la funzione `pb.fire(2,'Y')`, lo schema risultante sarà il seguente:

```
========
|RRBY G|
|R RR G|
|Y Y   |
|      |
|      |
|      |
|      |
|      |
|      |
========
```

Le bolle non possono attaccarsi fuori dello schema. Quindi una chiamata che dovesse attaccare una bolla sotto il bordo inferiore dello schema non ha effetto. La funzione deve essere concatenabile, quindi deve essere possibile scrivere `pb.fire(2,'Y').fire(1,'B')`. Se uno degli input non è valido, la funzione non ha effetto.

✓ **`(int)pb;`**

Operatore di conversione a intero per il tipo `PuzzleBobble`. Restituisce l'altezza del blocco di bolle nello schema, cioè il numero di righe che vanno dal bordo superiore fino all'ultima riga in basso che contiene almeno una bolla. Per esempio, nello schema visualizzato sopra la funzione restituirà 3.

--- Metodi invocati nella SECONDA PARTE di main.cpp: ---

✓ **`pb.fire(i,color);`** ***(versione avanzata della precedente versione)***

Modificare il metodo `pb.fire()` di modo che, se la bolla appena attaccata forma file di bolle dello stesso colore orizzontali e/o verticali il cui numero complessivo di bolle (considerando sia l'eventuale fila verticale che l'eventuale fila orizzontale) è maggiore o uguale a 3, tali bolle scoppiano e spariscono dallo schema. (Nota che questo potrebbe rendere alcune bolle non più attaccate a niente. Tali bolle rimangono nello schema e NON devono sparire.) Per esempio, se nello schema visualizzato precedentemente viene chiamata la funzione `pb.fire(1,'R')`, lo schema risultante sarà il seguente:

```
========
|R BY G|
|     G|
|Y Y   |
|      |
|      |
|      |
|      |
|      |
|      |
|      |
========
```

Nota che 5 bolle rosse sono scoppiate e sparite dallo schema, mentre due bolle gialle sono rimaste non attaccate a niente. La bolla rossa in alto a sinistra non è scoppiata in quanto non faceva parte né della fila verticale né di quella orizzontale formate dalla bolla appena attaccata.

✓ **`pb.scroll();`**

Metodo che fa scorrere il bordo superiore dello schema di una riga verso il basso, inserendo nella riga più alta una barra rappresentata da caratteri `'='`. Di conseguenza, il numero di righe disponibili per le bolle diminuisce di uno. Per esempio, se nello schema visualizzato sopra viene chiamata la funzione `pb.scroll()`, lo schema risultante sarà il seguente:

```
========
|======|
|R BY G|
|     G|
|Y Y   |
|      |
|      |
|      |
|      |
|      |
|      |
========
```

Le bolle non possono uscire dal limite inferiore dello schema. Quindi una chiamata che dovesse fare uscire alcune bolle non ha effetto. L'operatore `(int)pb` deve considerare ogni barra come una fila di bolle, quindi lo schema visualizzato sopra ha altezza 4. La funzione deve essere concatenabile.

✓ **`pb.compact();`**

Metodo che compatta ogni colonna di bolle verso l'alto, eliminando le caselle vuote tra bolla e bolla. Per esempio, se nello schema visualizzato sopra viene chiamata `pb.compact()`, lo schema risultante sarà:

```
========
|======|
|R BY G|
|Y Y  G|
|      |
|      |
|      |
|      |
```

```
|     |
|     |
|     |
=======
```

Programma di test:

```
#include "compito.h"
#include <iostream>
using namespace std;

int main(){
   // PRIMA PARTE:
   cout << "--- PRIMA PARTE ---" << endl;
   cout << "Test costruttore" << endl;
   PuzzleBobble pb;
   cout << pb;

   cout << "Test funzione fire" << endl;
   pb.fire(0,'R').fire(1,'R').fire(0,'B').fire(2,'Y');
   pb.fire(3,'Y').fire(3,'Y').fire(0,'B').fire(3,'G');
   cout << pb;

   cout << "Test operatore int" << endl;
   cout << "Altezza: " << (int)pb << endl;

   // SECONDA PARTE:
   cout << "--- SECONDA PARTE ---" << endl;
   cout << "Test funzionalita' scoppio bolle" << endl;
   pb.fire(0,'B'); // scoppio verticale di 3 bolle
   pb.fire(0,'R'); // no scoppio
   pb.fire(5,'Y').fire(4,'Y'); // scoppio orizzontale di 4 bolle
   pb.fire(3,'G'); // no scoppio
   cout << pb;

   cout << "Test funzione scroll" << endl;
   pb.scroll().scroll();
   cout << pb;

   cout << "Test funzione compact" << endl;
   pb.compact();
   cout << pb;

   return 0;
}
```

Uscita del programma di test:

```
--- PRIMA PARTE ---
Test costruttore
========
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
|       |
========
Test funzione fire
```

```
========
|RRYY  |
|B  Y  |
|B  G  |
|      |
|      |
|      |
|      |
|      |
|      |
|      |
========
Test operatore int
Altezza: 3
--- SECONDA PARTE ---
Test funzionalita' scoppio bolle
========
|RR    |
|R  Y  |
|   G  |
|   G  |
|      |
|      |
|      |
|      |
|      |
|      |
========
Test funzione scroll
========
|======|
|======|
|RR    |
|R  Y  |
|   G  |
|   G  |
|      |
|      |
|      |
|      |
========
Test funzione compact
========
|======|
|======|
|RR Y  |
|R  G  |
|   G  |
|      |
|      |
|      |
|      |
|      |
========
```

Es. 12.2 – History

Una `History` rappresenta un insieme di eventi storici. Ogni evento storico è rappresentato da un anno e da una descrizione di al più 30 caratteri.

Implementare le seguenti operazioni che possono essere effettuate su una `History`.

--- PRIMA PARTE ---

✓ `History hist;`
Costruttore di default che inizializza una `History` vuota, in cui cioè non ci sono eventi.

✓ `cout << hist;`
Operatore di uscita per il tipo `History`. L'uscita ha la forma seguente:

```
-- HISTORY --
480 BC
Battle of Thermopylae
-----
460 BC
First Peloponnesian War
-----
218 BC
Second Punic War
-----
44 BC
Julius Caesar is assassinated
-----
64 AD
Much of Rome burns
-----
410 AD
Visigoths sack Rome
-----
476 AD
Last Roman emperor is deposed
-----
```

Notare che gli eventi sono stampati in ordine cronologico, dal più antico al più recente. Nel caso due eventi cadano nello stesso anno, sono stampati nell'ordine in cui sono stati registrati (vedi funzione `record`). Le date prima di Cristo sono seguite da "**BC**", mentre quelle dopo Cristo da "**AD**". Sotto ogni data è stampata la descrizione, e sotto la descrizione una riga di separazione "**-----**". Tutta la `History` è preceduta da una riga "**-- HISTORY --**".

✓ `hist.record(year, descr);`
Operazione che registra un nuovo evento storico su una `History`, accaduto nell'anno `year` e descritto da `descr`. Si noti che per convenzione storiografica, l'anno 0 non esiste e l'anno successivo al 1 BC è il 1 AD. Per semplicità, l'argomento `year` segue lo standard ISO 8601, che fa corrispondere gli anni prima di Cristo a numeri negativi o zero (`year=0` corrisponde al 1 BC, `year=-1` al 2 BC, e così via) e gli anni dopo Cristo a numeri positivi (`year=1` corrisponde al 1 AD, e così via). Se gli input non sono del formato giusto, la `History` rimane inalterata.

✓ `hist.forget(descr);`
Operazione che fa dimenticare ad una `History` l'evento storico descritto da `descr`. Nel caso due eventi abbiano la stessa descrizione, viene dimenticato quello più antico. Nel caso abbiano anche lo

stesso anno, viene dimenticato il primo ad essere stato registrato. Se l'input non è del formato giusto, la `History` rimane inalterata.

✓ `~History();`
Distruttore.

--- SECONDA PARTE ---

✓ `hist.longest_period();`
Operazione che restituisce l'intervallo di tempo più lungo tra due eventi storici consecutivi in una `History`, misurato in anni. Tra due eventi accaduti rispettivamente il 5 BC ed il 20 AD passano 24 anni. Se una `History` non ha almeno due eventi storici, non è possibile calcolare intervalli di tempo, quindi la funzione `longest_period` restituirà il valore speciale -1.

✓ `hist.forget(from_year, to_year);`
Operazione che fa dimenticare alla `History` tutti gli eventi storici accaduti dall'anno `from_year` all'anno `to_year` compresi. Gli argomenti `from_year` e `to_year` seguono lo standard ISO 8601 sopra descritto. Se gli input non sono validi, la `History` rimane inalterata.

✓ `create_alternative(hist1, fork_year, hist2);`
Funzione globale che alloca e restituisce un puntatore ad una nuova istanza di `History`, alternativa a `hist1`. La `History` alternativa è identica a `hist1` fino all'anno di biforcazione `fork_year` compreso, dopo il quale è identica alla `hist2`. Per esempio, se dalla `History` di cui sopra si calcola un'alternativa con anno di biforcazione 300 AD e `hist2` uguale a:

```
-- HISTORY -
100 AD
Irrelevant event before fork
-----
410 AD
Visigoths fail to sack Rome
-----
1969 AD
First Roman astronaut on Moon
-----
```

allora la `History` alternativa sarà la seguente:

```
-- HISTORY --
480 BC
Battle of Thermopylae
-----
460 BC
First Peloponnesian War
-----
218 BC
Second Punic War
-----
44 BC
Julius Caesar is assassinated
-----
64 AD
Much of Rome burns
-----
410 AD
Visigoths fail to sack Rome
-----
1969 AD
First Roman astronaut on Moon
-----
```

L'argomento `fork_year` segue lo standard ISO 8601 sopra descritto.

Programma di test:

```
#include <iostream>
#include "compito.h"

using namespace std;

int main()
{
    cout << "--- PRIMA PARTE ---" << endl;
    cout << "Test del costruttore:" << endl;
    History hist;
    cout << hist << endl;

    cout << "Test della record:" << endl;
    hist.record(503, "aaa");
    hist.record(599, "bbb");
    hist.record(-107, "ccc");
    hist.record(405, "ddd");
    hist.record(711, "eee");
    hist.record(902, "fff");
    cout << hist << endl;

    cout << "Test della forget:" << endl;
    hist.forget("aaa");
    hist.forget("ccc");
    cout << hist << endl;

    cout << "Test del distruttore:" << endl;
    {
        History hist2;
        hist2.record(500, "aaa");
        hist2.record(400, "bbb");
        hist2.record(600, "ccc");
    }
    cout << "(oggetto distrutto)" << endl;

    // SECONDA PARTE
    cout << endl << "--- SECONDA PARTE ---" << endl;
    cout << "Test longest_period:" << endl;
    cout << hist.longest_period() << endl;

    cout << "Test forget overloaded:" << endl;
    hist.forget(500, 750);
    cout << hist << endl;

    cout << "Test create_alternative:" << endl;
    History hist3;
    hist3.record(-75, "ggg");
    hist3.record(507, "hhh");
    hist3.record(753, "iii");
    hist3.record(821, "jjj");
    History *p_hist4 = create_alternative(hist, 450, hist3);
    cout << *p_hist4 << endl;
    delete p_hist4;

    return 0;
}
```

Uscita del programma di test:

```
--- PRIMA PARTE ---
Test del costruttore:
-- HISTORY --

Test della record:
-- HISTORY --
108 BC
ccc
-----
405 AD
ddd
-----
503 AD
aaa
-----
599 AD
bbb
-----
711 AD
eee
-----
902 AD
fff
-----

Test della forget:
-- HISTORY --
405 AD
ddd
-----
599 AD
bbb
-----
711 AD
eee
-----
902 AD
fff
-----

Test del distruttore:
(oggetto distrutto)

--- SECONDA PARTE ---
Test longest_period:
194
Test forget overloaded:
-- HISTORY --
405 AD
ddd
-----
902 AD
fff
-----

Test create_alternative:
-- HISTORY --
405 AD
ddd
-----
507 AD
```

```
hhh
-----
753 AD
iii
-----
821 AD
jjj
-----
```